# UNIVERSIDADE PRESBITERIANA MACKENZIE
## - Faculdade de Computação e Informática –

**Ciência da Computação**
**Compiladores – 06N**
**Prova 1 – 09 de outubro de 2020**
**Professor: Fabio Lubacheski**

**Orientações para elaboração da prova:**

Esta prova pode ser feita em **dupla** ou **individualmente**, basta que somente um dos integrantes entregue um **arquivo pdf** com as respostas das questões e com o seguinte cabeçalho no início do arquivo.

Nós,

Bruno Severo Camilo tia: 41781619
Natália Gama de Mattos tia: 31887201

declaramos que

todas as respostas são fruto de nosso próprio trabalho,
não copiamos respostas de colegas externos a dupla,
não disponibilizamos nossas respostas para colegas externos a dupla e
não realizamos quaisquer outras atividades desonestas para nos beneficiar ou prejudicar outros.

**1) (1,0 ponto)** O Que são palavras reservadas (=palavras chaves) ? Qual sua função nas linguagens de alto nível? Como é que o analisador léxico efetua o reconhecimento de palavras reservadas ?

Uma palavra reservada é uma palavra que não pode ser utilizada para outro propósito além do original dentro do programa, ou seja, é uma palavra reservada para uso da gramática da linguagem.

A função das palavras reservadas em linguagens de alto nível é representar uma instrução.

O analisador léxico lê o arquivo fonte, e separa os caracteres agrupado em atomos e cada atomo é comparado com as palavras reservadas internas, assim gerando uma saida para o analisador sintático.

**2) (1,0 ponto)** O que são compiladores *just-in-time* ? De um exemplo de linguagem de programação com essa característica.

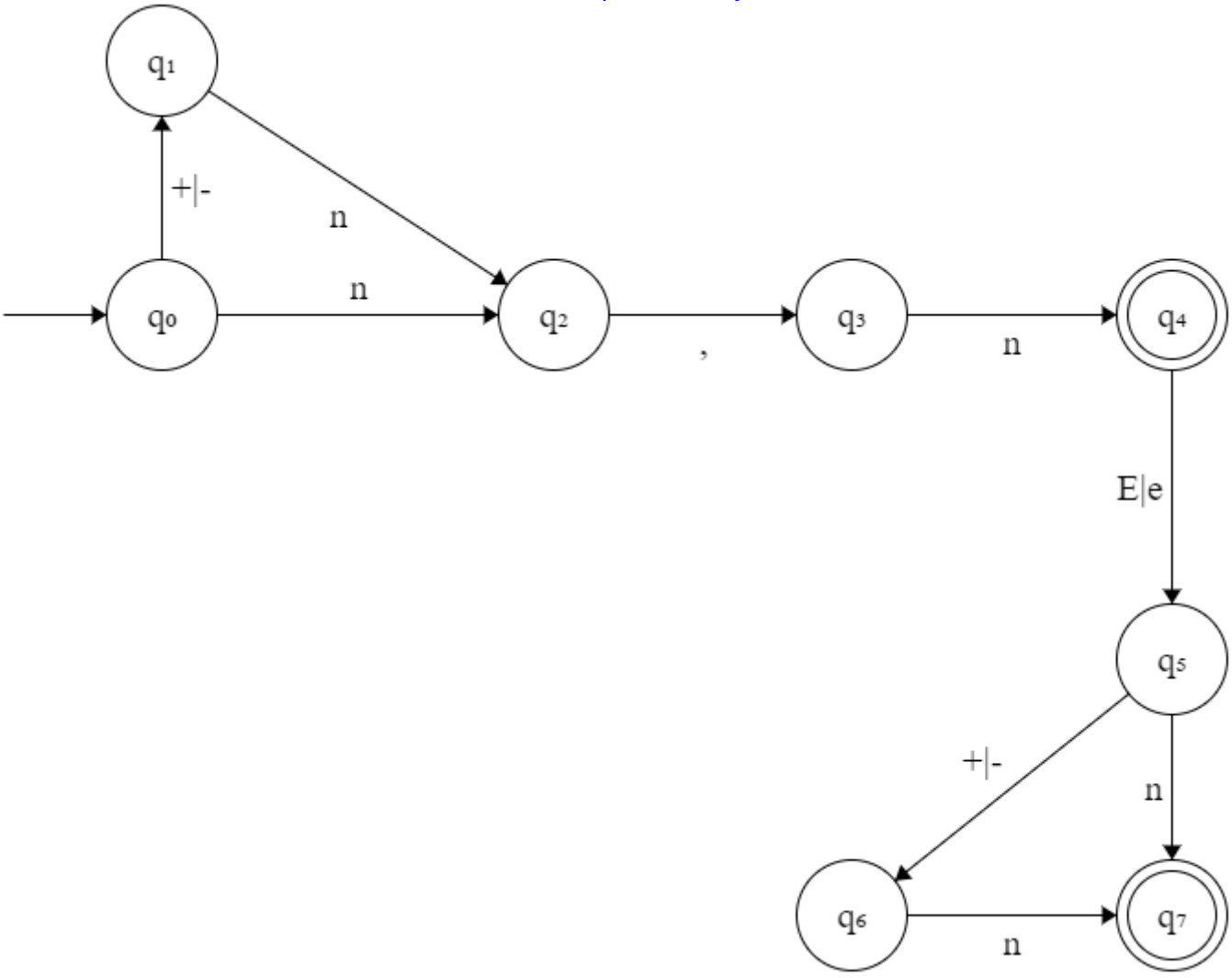
Compiladores just-in-time ou JIT são compiladores que são chamados no tempo da execução, ou em cada chamada de execução é feito um processo de compilação. O Java e o C# são bons exemplos de linguagens que utilizam o JIT.

**3) (2,5 pontos)** Considere a expressão regular para **constantes numéricas** de uma linguagem algorítmica com a seguinte notação abaixo:

`(+|-)?n,n((E|e)(+|-)?n)?`

onde
**n** é uma sequência de um ou mais dígitos, ou seja, $(0|1|2|3|4|5|6|7|8|9)^{+}$;
A notação **x?** significa que **x** é opcional, ou seja $(x|\varepsilon)$.

a) Construa um autômato para a expressão regular acima, para esse item você deve desenhar o diagrama de estados do autômato usando o site http://madebyevan.com/fsm/.




b) Baseado no autômato, implemente uma **função** que reconheça as constantes numéricas representadas pela expressão regular.

```c
int reconheceExpressao(char* input){
  int i = 0;
  q0:
    if(input[i] == '+' || input[i] == '-')
      i++;
      goto q1;
    else if(input[i] == 'n')
      i++;
      goto q2;
    else
      return 0;
```

```
  q1:
    if(input[i] == 'n')
      i++;
      goto q2;
    else
      return 0;
  q2:
    if(input[i] == ',')
      i++;
      goto q3;
    else
      return 0;
  q3:
    if(input[i] == 'n')
      i++;
      goto q4;
    else
      return 0;
  q4:
    if(input[i] == 'E' || input[i] == 'e')
      i++;
      goto q5;
    else if(input[i] == '\0')
      return 1;
    else
      return 0;
  q5:
    if(input[i] == '+' || input[i] == '-')
      i++;
      goto q6;
    else if(input[i] == 'n')
      i++;
      goto q7;
    else
      return 0;

  q6:
    if(input[i] == 'n')
      i++;
      goto q7;
    else
      return 0;
  q7:
    if(input[i] == '\0')
      return 1;
    else
      return 0;
}
```

**4) (2,0 pontos)** Considere a gramática na notação BNF, com a símbolo inicial S.

S ::= iEtS | iEtSeS | a
E ::= b

Esta gramática é ambígua? explique porque a gramática é ambígua.

Sim, pois uma palavra pode ser gerada de mais de uma forma na gramatica.
Ex: palavra ibtibtaea
iEtS::= ibtS ::= ibtiEtSeS ::= ibtibtSeS ::= ibtibtaea

iEtSeS::= ibtSeS ::= ibtiEtSeS ::= ibtibtSeS ::= ibtibtaea

Caso seja retire a ambiguidade da mesma de forma que ela continue gerando a mesma linguagem.

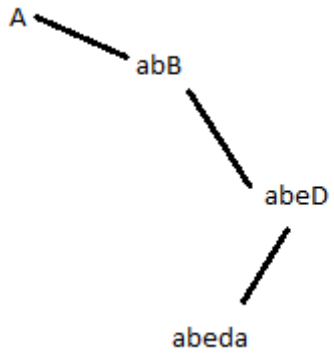
S ::= iEtSB | a
B ::= eS | λ
E :: = b

1) **(2,5 pontos)** Considere a gramática na notação BNF, com a símbolo inicial A.

A::= abB
B::= cC | eD
D::= da
C::= bC | λ

a) Apresente a árvore de derivação para cadeia abeda.

A::= abB ::=abeD ::= abeda.




b) Construa um analisador sintático descendente recursivo para essa gramática, considere que a cadeia que será testada já está armazenada na variável `*buffer` e chamada do analisador seria feita na função `main()` conforme abaixo.

```
char *buffer="abeda";

int main(void){
    printf("analisando %s\n",buffer);
    A();
    if( *buffer == '\x0')// fim de buffer
        printf("palavra aceita.");
    else
        printf("\nerro sintatico.");

    printf("\nfim de programa.\n");
    return 0;
}
```

```
char *buffer="abeda";

int A();
int B();
int C();
int D();

int A(){
  if(*buffer == 'a'){
    buffer++;
    if(*buffer == 'b'){
      buffer++;
      B();
```

```
    }
    else return 0;
  }
  else return 0;
}

int B(){
  if(*buffer == 'c'){
    buffer++;
    C();
  }
  else if (*buffer == 'e'){
    buffer++;
    D();
  }
  else return 0;
}

int C(){
  if(*buffer == 'b'){
    buffer++;
    C();
  }
  if(*buffer == ' '){
    buffer++;
  }
  else
    return 0;
}

int D(){
  if(*buffer == 'd'){
    buffer++;
    if(*buffer == 'a')
      buffer++;
    else return 0;
  }
  else return 0;
}

int main(){
printf("analisando %s\n",buffer);
A();
if( *buffer == '\x0')// fim de buffer printf("palavra aceita.");
  printf("palavra aceita.");
else
  printf("\nerro sintatico.");

printf("\nfim de programa.\n");
return 0;
}
```

**2) (1,0 ponto)** Considere a gramática

S ::= ABCDd
A ::= aA | λ
B ::= bC | λ
C ::= cD | λ
D ::= e

Apresente o FIRST de cada um dos não-terminais da gramática

S = {a,b,c,e}
A = {a}
B = {b}
C = {c}
D = {e}

**Boa Prova !**